Subject: Smoothing method in class Kalman
Posted by Anonymous Poster on Wed, 11 Mar 2009 10:35:19 GMT
View Forum Message <> Reply to Message

Hi everybody,

I am cleaning up genfit and trying to make things a bit tighter. One item which I would like to clean up is the Kalman class. In particular I would like to remove the smoothing method for the time being.
This method was implemented as shown in Fruehwirth and Regler and it had the following point to it: To get rid of the bias on the track parameters when you only do one fitting pass. However this was never tested and used by us, since it was replaced with the much simpler approach of several-pass-fitting. We agreed in dicussions that this solution is fine for use, since it gave good results.

That I want to remove this method does not mean that smoothing is going to be a problem in genfit. We just need to come up with a reason we need it and a choice for a solution. I have something in mind which I will briefly go over in my note I'm preparing.

In the moment the method is referenced in four methods, namely:
PndLheKalmanTask::Exec(), PndDchKalmanTask2::Exec(), PndDchKalmanTask::Exec(), and DemoKalmanTask::Exec()

I assume that Sebastian put it DemoKalmanTask and then it was just copied. In these methods smoothing is protected by a flag which is default false. Does anybody use this flag? If so, please tell me and we can discuss how to proceed. If it is not used I would kindly ask the Dch and lhetrack experts to remove those calls to Kalman::Smoothing(). I can do it myself if I get the write access.

I am looking forward to your (Dch, and Lhe) responses so I can continue my work!

Cheers, Christian

Subject: Re: Smoothing method in class Kalman
Posted by Aleksandra Wronska on Wed, 11 Mar 2009 11:08:41 GMT
View Forum Message <> Reply to Message

Hi Christian,

as for me, feel free to remove smoothing.

cheers,
ola

Subject: Re: Smoothing method in class Kalman
Posted by Anonymous Poster on Wed, 11 Mar 2009 11:12:40 GMT
View Forum Message <> Reply to Message

Hi,

sure thing. Cold you please remove it in your two tasks? I dont have write access.

Cheers, Christian

Subject: Re: Smoothing method in class Kalman (please remove calls)
Posted by Anonymous Poster on Fri, 13 Mar 2009 11:30:46 GMT
View Forum Message <> Reply to Message

Hi,

thanks Aleksandra for removing the lines. I would like to ask the LHE experts to remove theirs, so I can carry on.

Cheers, Christian

Subject: Re: Smoothing method in class Kalman (please remove calls)
Posted by StefanoSpataro on Fri, 13 Mar 2009 12:13:28 GMT
View Forum Message <> Reply to Message

Done.
In each case I am not so sure if removing the smoothing is a good idea.
If this is implemented, and if it works properly, it could stay in the code without any harm. Or not?

Subject: Re: Smoothing method in class Kalman (please remove calls)
Posted by Anonymous Poster on Fri, 13 Mar 2009 12:30:58 GMT
View Forum Message <> Reply to Message

Hi Stefano,

thanks for removing... I wouldnt say the smoothing worked. This kind of smoothing was to remove bias on the all the hits, but it was very clumsy in implementation. Well in a way it worked, it wasnt really wrong or anything, but we replaced its purpose by multiple pass fitting. We will come back to the issue for sure in the future, when we will determine how to get track parameters in between the track extremities without any loss of information. We can get these parameters even now, but not with the best covariance possible, yet. Let's come back to this in some weeks.

Cheers, Christian

Subject: Re: Smoothing method in class Kalman
Posted by Anonymous Poster on Fri, 13 Mar 2009 20:51:00 GMT
View Forum Message <> Reply to Message

Hi,

I have another small request. I had made the residualVector method of AbsRecoHit virtual. It is actually being overwritten in PndDchRecoHit. This should not be the case. I am sorry for the inconvenience. I want to slightly change the prototype.

In my system the code still compiles, and I am pretty sure it will on others. However this is dangerous and the definition should be removed from Dch.

Thanks, Christian

Subject: Re: Smoothing method in class Kalman
Posted by Aleksandra Wronska on Mon, 16 Mar 2009 11:03:06 GMT
View Forum Message <> Reply to Message

Dear Christian,

I am not sure that I got your point concerning the PndDchRecoHit::residualVector(...). PndDchRecoHit indeed needs a non-standard version of this function. I guess the problem was rather to change the arguments of this function in dch such, that they match the pattern of AbsRecoHit..? If so, it's done now. If not, I'll need you to be more specific.

cheers,
ola

P.S. Why is there this additional argument detPlane in residualVector(...)? residualVector is called for a RecoHit, and this in turn is defined together with its detection plane. Why do we need to provide this extra information?

Subject: Re: Smoothing method in class Kalman
Posted by Anonymous Poster on Mon, 16 Mar 2009 12:45:29 GMT
View Forum Message <> Reply to Message

Hi Ola,

ah I wasnt aware that your hit needs a custom residualVector implementation. That is not a problem though, I will make the residualVector method virtual again, immediately, such that your overwritten method is used. I am not sure about the functionality of C++ in this case, but I guess that if you provide your own implementation with the same prototype, but in the super class the thing was not virtual, if the method is called on a super-class pointer, the super-class implementation is called and not yours. We do want to avoid that! As I said, I will make it virtual again now.

What I will write next will be complicated, but it will fully answer your question. You will have much more documentation on Genfit soon, and then it will be easier to understand what I am talking about. But I will give you a full explanation here anyway. Afterall, the code is not a lot to look through...

For an explanation why I changed the prototype to contain the DetPlane and also why I made

the AbsTrackRep pointer to point to a const AbsTrackRep:
In residual vector you need the DetPlane of the hit. In general this DetPlane is not fixed for the hit, but it depends on the track. This is implemented in the TPC through SpacePointHitPolicy. This means that if you would not give a DetPlane to the residualVector method, it would have to calculate it. This is the way it worked before. However, this calculation is expensive in computing time, and it has been done before, so we dont want to do it again. I made the AbsTrackRep pointer const, so extrapolate (which before was called through AbsTrackRep::getVirtualDetectorPlane over the SpacePointHitPolicy) is not possible anymore. There is another reason why I want to prohibit extrapolate calls inside any method of AbsRecoHit besides extrapolateToPoca (which is called by getVirtualDetectorPlane): I do multi-pass fitting in Kalman. When I reached the last point and I turn around, this first hit now after turning around does not require any extrapolation, since I am already there. These kind of null-extrapolations need to be avoided since Geane is instable for them, right now. So I dont do the extrapolation in these cases (look inside Kalman::ProcessHit()), but then I need to make sure that the residualVector method does also not do any extrapolations.

I commit the change for virtual residualVector in a minute.

Cheers, Christian

Subject: inheritance in C++ important question
Posted by Anonymous Poster on Mon, 16 Mar 2009 13:09:28 GMT
View Forum Message <> Reply to Message

Hi,

what I stated in my last reply in this thread about virtual in C++ turns out to be true. Please compile and run the following snipplet

```
//---------------
#include<iostream>

class A{
public:
  virtual void doit(){std::cout<<"A"<<std::endl;}
};
class B : public A{
public:
  void doit(){std::cout<<"B"<<std::endl;}
};
class C{
public:
  void doit(){std::cout<<"C"<<std::endl;}
};
class D : public C{
public:
  void doit(){std::cout<<"D"<<std::endl;}
};
```

```
int main(){
  A* x = new B;
  C* y = new D;
  x->doit();
  y->doit();
  ((D*)y)->doit();
}
//---------------
```

It prints
B
C
D
and the compiler gives no warning, also not with -Wall -pedantic. Does anybody know how I can force the compiler to not allow the overwriting at all, like in the case of class D where I (one could mistakenly think) overwrite the method?

Please help a C++ fool!

Cheers, Christian

Subject: Re: inheritance in C++ important question
Posted by StefanoSpataro on Mon, 16 Mar 2009 13:27:12 GMT
View Forum Message <> Reply to Message

Hi,
sorry but I have not understood well what is your concern.
Of course with ((D*)y) you force to use the functions of the base class "D" for y, and not the ones of the inherited C. This should be always taken into account when using "brute force" casting, and the compiler does not complain. I think this is a feature of C++ and it cannot be avoided at all.

For the RecoHit businnes maybe one should move the function from the daughter class to the mother one (if possible, maybe using some flags), in order to avoid problems. But I have not yet digged inside the code so I am not completely aware on how the problem rises in the execution.

Subject: Re: inheritance in C++ important question
Posted by Anonymous Poster on Mon, 16 Mar 2009 13:48:40 GMT
View Forum Message <> Reply to Message

Hi Stefano,

thanks for your interest. My main interst lies in the second doit call (not so much the third, which I think is obvious). But the second call does something confusing. Since in C the doit method is not virtual, in fact the call to y->doit() (y is a C*) calls the code in C, and not the iherited D class. This is clear to me, since there is no vtable for that method.

But: This is dangerous. When I write a method in the super class I would like to prohibit people to reimplement this method, because they mistakenly will assume that their code will be called. However if the object is accessed through a super class pointer, like in my example C* y = new D; the code you implement in D will never be called through the super class pointers.

Any clearer? Please keep asking what I didnt make clear. This one is really important for all object oriente code. I naively assumed that the compiler would prohibit you from doing that!

Cheers, Christian

Subject: Re: inheritance in C++ important question
Posted by Aleksandra Wronska on Mon, 16 Mar 2009 14:39:11 GMT
View Forum Message <> Reply to Message

Dear Christian,

the result of your example is what I would expect. This is how the virtuality mechanism works, isn't it? So if you want always to have the functions of daughter classes executed, even after casting to mother class, why don't you make these functions virtual? I am not getting your point...

Concerning the PndSttRecoHit and PndDchRecoHit2, there was a discussion how to solve it in the forum some weeks ago. The hits are very similar and originally I even derived my dch hit from stt one, but for tidyness and dependency reasons we separated it, keeping the common WirepointHitPoilicy in the genfit directory.

However, there are tiny but important differences in the detPlane(...) functions of those classes, so these functions have to be reimplemented in the dch and stt reco hits. As for the getDetPlane(...), I think we can remove it from our classes (Lia and Susanna, correct me if I am wrong). Is this what you meant?

cheers,
ola
